Discussion Section Week 1

• Intro

• Course Project Information

• Constraint Satisfaction Problems
  – Sudoku
  – Backtracking Search Example
  – Heuristics for guiding Search Example
Intro

• Teaching Assistant
  – Junkyu Lee (June Queue Lee)
  – Office Hour
    • Friday 11:00 AM ~ 12:00 PM
    • DBH 4099

• Reader
  – Minhaeng Lee (Min Heng Lee)
  – Office Hour
    • Thursday 2:00 PM ~ 3:00 PM
    • DBH 4219
Course Project Information

- Fri., 15 Jan., 11:59pm: Project Team Formation
- Sun., 24 Jan., 11:59pm: Project **Problem** Generator
- Sun., 31 Jan., 11:59pm: Project **Backtracking** Search
- Sun., 14 Feb., 11:59pm: Project **Forward Checking**
- Sun., 21 Feb., 11:59pm: Project **Arc Consistency**
- Sun., 28 Feb., 11:59pm: Project **MRV & DH Heuristic**
- Sun., 6 Mar., 11:59pm: Project **LCV** Heuristic
- Sun., 13 Mar., 11:59pm: Final Project

You will lose 10% of your Project grade for every day or fraction thereof it is late.
Course Project Information

• Fri., 15 Jan., 11:59pm: Project Team Formation
  
  – How Many Members?

  – We will post a google doc next week on EEE message board
You Will Be Expected to Know

• Basic definitions (section 6.1)
  – What is a CSP?

• Backtracking search for CSPs (6.3)

• Variable ordering or selection (6.3.1)
  – Minimum Remaining Values (MRV) heuristic
  – Degree Heuristic (DH) (to unassigned variables)

• Value ordering or selection (6.3.1)
  – Least constraining value (LCV) heuristic
What is CSP?

- Task
- Model
What is CSP?

• Task/goal for solving CSP
  – Given a set of constraints,
    • Find a solution that satisfy all constraints
    • Find all solutions that satisfy all constraints
    • Count number of solutions
    • ...

What is CSP?

• How to model/express CSP problems?
  – variable and its domain

  – constraints, relations, functions
    • allowed (partial) combinations of variable values
**Constraint Satisfaction Problems**

- **What is a CSP?**
  - Finite set of variables $X_1, X_2, \ldots, X_n$
  - Nonempty domain of possible values for each variable $D_1, D_2, \ldots, D_n$
  - Finite set of constraints $C_1, C_2, \ldots, C_m$
    - Each constraint $C_i$ limits the values that variables can take,
    - e.g., $X_1 \neq X_2$
  - Each constraint $C_i$ is a pair <scope, relation>
    - Scope = Tuple of variables that participate in the constraint.
    - Relation = List of allowed combinations of variable values.
      - May be an explicit list of allowed combinations.
      - May be an abstract relation allowing membership testing and listing.

- **CSP benefits**
  - Standard representation pattern
  - Generic goal and successor functions
  - Generic heuristics (no domain specific expertise).
Sudoku
Example: Sudoku (constraint propagation)

Each row, column and major block must be all different.

“Well posed” if it has unique solution: 27 constraints

Variables: 81 slots
Domains = \{1,2,3,4,5,6,7,8,9\}
Constraints: 27 not-equal
Sudoku
(inference)

Each row, column and major block must be all different

“Well posed” if it has unique solution
Backtracking search

• Similar to Depth-first search
  – At each level, picks a single variable to explore
  – Iterates over the domain values of that variable

• Generates kids one at a time, one per value

• Backtracks when a variable has no legal values left

• Uninformed algorithm
  – No good general performance
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom
Backtracking search

• Expand *deepest* unexpanded node
• Generate *only one* child at a time.
• *Goal-Test* when inserted.
  – For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

- Expand *deepest* unexpanded node
- Generate *only one* child at a time.
- *Goal-Test* when inserted.
  - For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

• Expand \textit{deepest} unexpanded node
• Generate \textit{only one} child at a time.
• \textit{Goal-Test} when inserted.
  – For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search

• Expand *deepest* unexpanded node
• Generate *only one* child at a time.
• *Goal-Test* when inserted.
  – For CSP, Goal-test at bottom

Future= green dotted circles
Frontier=white nodes
Expanded/active=gray nodes
Forgotten/reclaimed= black nodes
Backtracking search (Figure 6.5)

function BACKTRACKING-SEARCH(csp) return a solution or failure
return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
        add {var=value} to assignment
        result ← RECURSIVE-BACKTRACKING(assignment, csp)
        if result ≠ failure then return result
        remove {var=value} from assignment
    return failure
Depth First Search

Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

A

B
D
H

C

E
I
J

F
K
L

G
N
O
Depth-first search

Expand deepest unexpanded node

Implementation:

\[ \text{fringe} = \text{LIFO queue, i.e., put successors at front} \]
Depth-first search

Expand deepest unexpanded node

**Implementation:**

\[ \text{fringe} = \text{LIFO queue, i.e., put successors at front} \]
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front
Depth-first search

Expand deepest unexpanded node

Implementation:

\[ \text{fringe} = \text{LIFO queue, i.e., put successors at front} \]
Depth-first search

Expand deepest unexpanded node

Implementation:

fringe = LIFO queue, i.e., put successors at front

![Tree diagram](image)
Depth-first search

Expand deepest unexpanded node

**Implementation:**

\[fringe = \text{LIFO queue}, \text{i.e., put successors at front}\]
Improving CSP efficiency

• Previous improvements on uninformed search
  → introduce heuristics

• For CSPS, general-purpose methods can give large gains in speed, e.g.,
  – Which variable should be assigned next?
  – In what order should its values be tried?
  – Can we detect inevitable failure early?
  – Can we take advantage of problem structure?

Note: CSPs are somewhat generic in their formulation, and so the heuristics are more general compared to methods in Chapter 4
Heuristic

• Selecting Variable
  – Minimum remaining values (MRV)
    • choose variable with the fewest legal moves
  – Degree heuristic for next variable
    • select variable that is involved in the largest number of constraints on other unassigned variables
    • useful as a tie breaker after MRV.

• Selecting Value
  – Least constraining value (LCV)
    • given a variable choose the least constraining value
Backtracking search (Figure 6.5)

function BACKTRACKING-SEARCH(csp) return a solution or failure
return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
if assignment is complete then return assignment
var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
        add {var=value} to assignment
        result ← RRECURSIVE-BACTRACKING(assignment, csp)
        if result ≠ failure then return result
    remove {var=value} from assignment
return failure
CSP example: Map coloring problem

Variables: WA, NT, Q, NSW, V, SA, T
Domains: $D_i = \{\text{red}, \text{green}, \text{blue}\}$
Constraints: adjacent regions must have different colors.
  • E.g. WA $\neq$ NT
CSP example: Map coloring solution

- A solution is:
  - A complete and consistent assignment.
  - All variables assigned, all constraints satisfied.

- E.g., \{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green\}
Minimum remaining values (MRV) for next variable

\[ \text{var} \leftarrow \text{SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)} \]

- A.k.a. most constrained variable heuristic

- *Heuristic Rule*: choose variable with the fewest legal moves
  - e.g., will immediately detect failure if X has no legal values
Degree heuristic for next variable

- **Heuristic Rule**: select variable that is involved in the largest number of constraints on other unassigned variables.

- Degree heuristic can be useful as a tie breaker after MRV.

- *In what order should a variable’s values be tried?*
function BACKTRACKING-SEARCH(csp) return a solution or failure
  return RECURSIVE-BACKTRACKING({}, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) return a solution or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment according to CONSTRAINTS[csp] then
      add {var=value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var=value} from assignment
  return failure
Least constraining value (LCV) for next value

- Least constraining value heuristic

- Heuristic Rule: given a variable choose the least constraining value
  - leaves the maximum flexibility for subsequent variable assignments
Minimum remaining values (MRV) vs. Least constraining value (LCV)

• Why do we want the MRV (minimum values, most constraining) for variable selection --- but the LCV (maximum values, least constraining) for value selection?

• Isn’t there a contradiction here?

• MRV for variable selection to reduces the branching factor.
  – Smaller branching factors lead to faster search.
  – Hopefully, when we get to variables with currently many values, constraint propagation (next lecture) will have removed some of their values and they’ll have small branching factors by then too.

• LCV for value selection increases the chance of early success.
  – If we are going to fail at this node, then we have to examine every value anyway, and their order makes no difference at all.
  – If we are going to succeed, then the earlier we succeed the sooner we can stop searching, so we want to succeed early.
  – LCV rules out the fewest possible solutions below this node, so we have the most chances for early success.